AI for TSP Competition
Problem statement

Laurens Bliek[2], Tom Catshoek[1], Paulo de Oliveira da Costa[2], Reza Refaei Afshar[2], Daniël Vos[1], Sicco Verwer[1], Yingqian Zhang[2]

[1] Delft University of Technology, Netherlands
[2] Eindhoven University of Technology, Netherlands

1 Introduction

The goal of this competition is to design AI methods to solve logistics problems such as variants of the traveling salesman problem (TSP). In such problems, the goal is to find a route through a network that maximizes some reward. There are two tracks that each require knowledge of a different subfield of AI:

- Track 1: online supervised learning / surrogate models. Given one instance, previously tried routes, and the reward for those routes, the goal is to learn a model that can predict the reward for a new route. Then an optimizer finds the route that gives the best reward according to that model, and that route is evaluated, giving a new data point. Then the model is updated, and this iterative procedure continues for a fixed number of steps. Over time, the model becomes more accurate, giving better and better routes. This procedure is used in surrogate-based algorithms such as Bayesian optimization [Shahriari *et al.*, 2016].
- Track 2: reinforcement learning. We consider an environment (simulator) that can generate multiple instances from a set of instances $\mathcal{I}$ following the same distribution and expects as output (partial) solutions containing the order at which the nodes should be visited. The simulator returns general instance features and the time-dependent cost for traversing the last edge in a given solution. The goal is to minimize the cost of the total path over multiple samples of selected test instances. This procedure is related to Reinforcement Learning and Neural Combinatorial Optimization [Bello *et al.*, 2017].

2 Problem description

The code for the two tracks can be found at https://github.com/paulorocosta/ai-for-tsp-competition. Both tracks will look at the time-dependent orienteering problem with stochastic weights and time windows (TD-OPSWTW) [Verbeeck *et al.*, 2016], which is a problem similar to the traveling salesman problem (TSP) where nodes need to be visited in a certain order. The stochastic and time-dependent constraints and rewards make this problem particularly difficult to solve with traditional solvers. For this reason, recently AI methods have been applied to similar problems in order to overcome the limitations of OR solvers with machine learning [Kool *et al.*, 2018; Bliek *et al.*, 2020].

In the TSP the goal is to find the tour with the smallest cost visiting all locations (customers) in a network exactly once. However, in practical applications, one rarely knows all the travel costs between locations precisely. Moreover, there could be specific time windows at which customers need to be served, and specific customers can be more valuable than others. Lastly, the salesman is often constrained by a maximum capacity or travel time, representing a limiting factor in the number of nodes that can be visited.

In this competition, we consider a more realistic version of the classical TSP, i.e., the TD-OPSWTW. In this formulation, the stochastic travel times between locations are only revealed as the salesman travels in the network. The salesman starts from a depot and must return to the depot at the end of the tour. Moreover, each node (customer) in the network has its prize, representing how important it is to visit a given customer on a tour. Each node has associated time windows. We consider that a salesman may arrive earlier at a node without compromising its prize, but the salesman has to wait until the opening times to serve the customer. Lastly, the salesman must not violate a total travel time budget while collecting prizes in the network. The goal is to collect the most prizes in the network while respecting the time windows and the total travel time of a tour allowed to the salesman.

2.1 Problem instances

A set of problem instances are generated and provided for the participants. Each problem instance contains n nodes in 2D space. Each node has a x-coordinate, a y-coordinate, lower and upper bounds of its time window, prizes and the maximum tour time. The time windows determine the desired period of visiting a particular node. If the time of visiting a node is within the time window (or before), the prize is collected.

The problem instances are generated in three steps. First, the coordinates of the nodes are generated randomly between a lower and an upper bound. The location of the nodes are fixed; however, the distances between the nodes are noisy and they might be different in different runs.

Second, the time window of each node is generated around the time of visiting that node in second nearest neighbor tour. In more detail, let t_i be the time of visiting node i in the second nearest neighbor tour. The left side of the time window

CUSTNO	XCOORD	YCOORD	TW_LOW	TW_HIGH	PRIZE	MAX_T
1	47.0	24.0	0	285	0.0	256
2	38.0	15.0	102	198	0.19	256
3	53.0	49.0	9	52	0.38	256
4	116.0	23.0	30	137	1.0	256

Table 1: A sample problem instance with 4 nodes.

is a randomly generated number between $t_i - w$ and t_i where w is an arbitrary integer. Similarly, the right side of the time window is a random number between t_i and $t_i + w$.

Third, the prize of each node is determined according to the L_2 distances between the nodes and depot. In order to prevent infinite loops which can gain very large total prize, each problem instance has a max length that determines the maximum length of a solution. All solutions longer than max length are penalized.

As a simple example, assume that there are 4 nodes that each one has a time window and a prize, and let us ignore that travel times between nodes are stochastic. An illustration of this example is shown in Table 1. Each row of this table corresponds to a particular node and the columns are defined as follows: CUSTNO is an integer identifier for the nodes. XCOORD and YCOORD are the coordinate of a node. TW_LOW and TW_HIGH are the left side and right side of the time window respectively. Finally, PRIZE is the prize of each node. Let the max length MAX_T be 256. A possible tour for this example is $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$. Total length of this tour using rounded distances between the nodes is 187 and the time of visiting nodes 1, 2, 3 and 4 are 0, 13, 50 and 118 respectively. At time 13, an agent visits node 2; however, it needs to wait until time 102 in order to collect the prize of this node. If the agent leaves node 2 after collecting its prize, it gets to node 3 after closing its time window. Therefore, the agent misses the prize of node 3. Then, it can get to node 4 within its time window and collect its prize. Therefore, the total prize of this tour is 1.19.

2.2 Differences with TSP

To summarize, the main differences between the problem in this competition and the traveling salesman problem are:

- Not all nodes need to be visited: it is allowed to never visit some nodes.
- Visiting a node after the node's opening time and before its closing time gives a reward.
- Visiting a node after its closing time gives a penalty (negative reward).
- When visiting a node before its opening time, the agent has to wait until the node opens.
- The time it takes to travel from one node to the other is stochastic.
- The travel times do not directly appear in the objective function, the only thing that matters is the reward.

3 Track 1

Contact: Laurens Bliek, l.bliek@tue.nl

The goal of track 1 is to solve an optimization problem related to one instance of the TD-OPSWTW problem, finding the route that maximizes the reward. The reward of a route can be represented as a black-box function $f(\mathbf{s}, i)$, taking as input the instance i and a route $\mathbf{s}$. The optimization problem is then denoted as:

$$\mathbf{s}^* = \underset{\mathbf{s}}{\operatorname{argmax}}\, E[f(\mathbf{s}, i)] \tag{1}$$

for a given instance i. We use the expected value because the simulator is stochastic: it can give different rewards even if the same route is evaluated multiple times. The expected value for a route $\mathbf{s}$ is approximated by evaluating $f(\mathbf{s}, i)$ for that route $10\,000$ times and calculating the average reward. This computation takes multiple seconds on standard hardware. Therefore, the problem can be seen as an expensive optimization problem. Surrogate-based optimization methods such as Bayesian optimization [Shahriari *et al.*, 2016], which approximate the expensive objective using online supervised learning, are known to perform well on this type of problems.

The route $\mathbf{s}$ indicates in which order to visit the nodes in the network. It has to take on the specific form $\mathbf{s} = [1, s_1, \ldots, s_n]$, with n the number of nodes and $s_1, \ldots, s_n$ containing all integers from 1 to n. This means that the number 1 will appear twice in the solution. As this number indicates the starting node, it means that the route consists of starting from the starting node, visiting any number of nodes, then returning to the starting node at some point. Any nodes that appear in the route after returning to the starting node are ignored.

3.1 Surrogate-based optimization

The problem from this track can in theory be tackled with any black-box optimization technique. However, the number of simulator calls is in practice limited due to the time it takes to calculate the average over all the Monte Carlo samples. For this reason, surrogate-based algorithms are particularly useful. These algorithms approximate the black-box function f in every iteration with a surrogate model g (the online learning problem), then optimize g instead (the optimization problem). Both the results of learning and optimization become better with each simulator call as more data becomes available. The problem is split into two sub-problems that are solved every time a route is given as input to the simulator:

1. Given the paths tried up until now and their corresponding rewards, learn a model that can predict for any new path how promising that path would be.
2. Optimize the model of the previous step to suggest the most promising path to try next. Then this path is given as input to the simulator.

The first step can be seen as an online learning problem, where new data comes in at every iteration and rewards need to be predicted. It also corresponds to the concept of an acquisition function in Bayesian optimization. In step 2, standard optimization methods can be used.

3.2 Baseline

As a baseline method, we provide an implementation of a standard Bayesian optimization algorithm using Gaussian processes [Shahriari *et al.*, 2016]. For this implementation, we use the `bayesian-optimization` Python package[1], after transforming the input space using the approach in [Bliek *et al.*, 2020] and rounding solutions to the nearest integer.

4 Track 2

Contact: Paulo da Costa, p.r.d.oliveira.da.costa@tue.nl

In this track we consider Reinforcement Learning (RL) methods. In the Reinforcement Learning (RL) track, we are interested in a policy π mapping states to action probabilities. A policy in the TD-OPSWTW selects the next node to be visited given a sequence of previously visited nodes. Note that to cope with the stochastic travel times, a policy must be adaptive. Therefore, a policy needs to consider the instance information to construct tours dynamically that respect the time windows of nodes and the total tour time allowed for the instance. Note that unlike Track 1, we are interested in general policies applicable to any instance of the TD-OPSWTW in the training distribution.

More formally, we adopt a standard Markov Decision Process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(\mathbf{s}'|\mathbf{s}, a)$ is the transition distribution after taking action a at state $\mathbf{s}$, $r(\mathbf{s}, a)$ is the reward function. Where the state space is composed of partial tours (or a node), the action space is comprised of the remaining nodes to be visited and the rewards are the sum of prizes and penalties collected at each step. Thus the main objective is to find a policy π^* such that

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \left[\sum_{i=0}^{n-1} r(s_{0:i}, a_i) \right]. \tag{2}$$

Note that for simplicity we assume that we always start from the depot, i.e., $s_0 = 1$. Figure 1 shows an example of a next node visitation decision that has to be made by a policy visiting $n = 6$ nodes.

In the figure, a policy has visited nodes 1 (depot) and 6 with travel time $t_{1,6}$ revealed after visiting node 6. Note that $t_{i,j} \in \mathbb{N}$, $\forall i, j \in \{1, \ldots, n\}$ are sampled from a r.v. $\mathbb{T}_{i,j}$, i.e., $t_{i,j} \sim \mathbb{T}_{i,j}$. At this current decision epoch, the policy has to choose the next node to visit. The prizes $\{p_i \in \mathbb{R}\}_{i=1}^n$ and time window bounds $\{(l_i, u_i) \in \mathbb{N}^2\}_{i=1}^n$ are known and given in the instance, as well as the maximum allowed tour time $T \in \mathbb{N}$. The decision should consider the prizes of each node, the time windows, and the total remaining travel time when selecting the next node (in this case, node 3).

[1]https://github.com/fmfn/BayesianOptimization

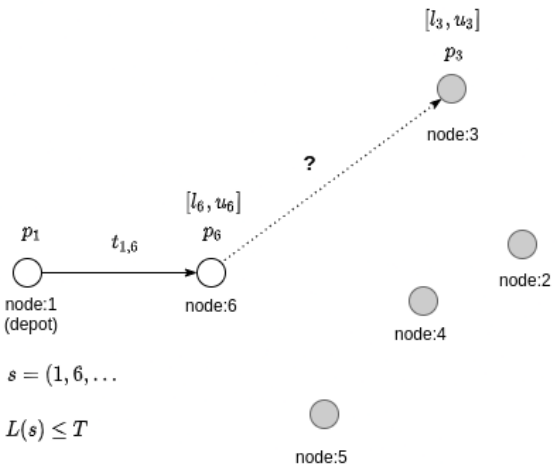

Figure 1: A policy solving the TD-OPSWTW dynamically.

We consider two constraints in the problem, both incurring *penalties* $\{e_i \in \mathbb{R}\}_{i=1}^n$. To avoid penalties and achieve a feasible solution, a policy needs to visit nodes respecting the upper bounds of the time windows, i.e., it can violate the lower bounds and arrive early without penalties. It should also respect the maximum tour time, i.e., $L(\mathbf{s}) \leq T$ where $\mathbf{s}$ is a partial solution up until $n' \leq n$ and $L(\mathbf{s}) = \sum_{i=1}^{n'} t_{s_{i-1}, s_i}$. Note that the reward after taking action a_i is given by $r(s_{0:i}, a_i) = p_{a_i} + e_{a_i}$.

Moreover, when a policy decides to arrive early at a node, the travel time gets shifted to the beginning of the time window. For example, if the travel time between the depot (node 1) and node 6 is lower than l_6, the salesman has to wait until l_6 to depart from that node. This information becomes available as soon as the salesman arrives at node 6. Lastly, a policy must always return to the depot, and this travel time is also included in the maximum allowed tour time.

Please check the details about the implementation in https://github.com/paulorocosta/ai-for-tsp-competition.

4.1 Baseline

We provide a baseline to the RL track based on [Bello *et al.*, 2017]. Note that this approach is not adaptive and does not perform well in the given task. This baseline is just a reference as to how RL can be used. Moreover, it only uses the coordinates and prizes to make decisions on complete tours.

5 Submission format

When it comes to submitting your results, you upload two files: the output of your algorithm and a ZIP file containing your code (for verifying the winners). The expected output of your algorithm differs between the two tracks:

- Surrogate models track: a file with `.out` extension containing the nodes to be visited separated by newlines. Should start with the number '1' and needs to contain '1' twice. See example below.
- Reinforcement learning track: a JSON file containing 'instance_names' (str). For each instance, there should

be a 'seed' (int), a 'nodes' (int), followed by N 'tours' (array of ints of size $n+1$), where N < 1000. See example below.

5.1 Surrogate models output example

File: `example.out`

```
1
3
2
5
1
4
```

5.2 Reinforcement learning output example

File: `example.json`

```
{"instance0001": {
    "nodes": 5,
    "seed": 12345,
    "tours": {
        "tour001": [1, 2, 3, 1, 5, 4],
        "tour002": [1, 5, 4, 2, 1, 3],
        }},
 "instance0002": {
    "nodes": 4,
    "seed": 33333,
    "tours":{
        "tour001": [1, 2, 1, 3, 4],
        "tour002": [1, 3, 1, 2, 4]
    }}
}
```

6 Evaluation/Scoring

Please refer to the competition repository.

7 Sponsors and prize money

This competition is sponsored by Ortec and Vanderlande. Please see https://www.tspcompetition.com/competition for information on the prize money.

8 Source Code

For detailed explanation of the code, please see the provided repository [2].

References

[Bello *et al.*, 2017] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *ArXiv*, abs/1611.09940, 2017.

[Bliek *et al.*, 2020] Laurens Bliek, Sicco Verwer, and Mathijs de Weerdt. Black-box combinatorial optimization using models with integer-valued minima. *Annals of Mathematics and Artificial Intelligence*, pages 1–15, 2020.

[Kool *et al.*, 2018] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

[Shahriari *et al.*, 2016] B. Shahriari, Kevin Swersky, Ziyu Wang, R. Adams, and N. D. Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104:148–175, 2016.

[Verbeeck *et al.*, 2016] C Verbeeck, Pieter Vansteenwegen, and E-H Aghezzaf. Solving the stochastic time-dependent orienteering problem with time windows. *European Journal of Operational Research*, 255(3):699–718, 2016.

[2] https://github.com/paulorocosta/ai-for-tsp-competition